Workload Clustering for Increasing Energy Savings on Embedded MPSoCs

S. H. K. Narayanan, O. Ozturk, M. Kandemir
Dept of CSE
The Pennsylvania State University
{snarayan,ozturk,kandemir}@cse.psu.edu

M. Karakoy
180 Queen's Gate
Imperial College
mk22@doc.ic.ac.uk

Abstract

Voltage/frequency scaling and processor low-power modes (i.e., processor shut-down) are two important mechanisms used for reducing energy consumption in embedded MPSoCs. While a unified scheme that combines these two mechanisms can achieve significant savings in some cases, such an approach is limited by the code parallelization strategy employed. In this paper, we propose a novel, integer linear programming (ILP) based workload clustering strategy across parallel processors, oriented towards maximizing the number of idle processors without impacting original execution times. These idle processors can then be switched to a low power mode to maximize energy savings, whereas the remaining ones can make use of voltage/frequency scaling. In order to check whether this approach brings any energy benefits over the pure voltage scaling based, pure processor shut-down based, or a simple unified scheme, we implemented four different approaches and tested them using a set of eight array/loop-intensive embedded applications. Our simulation-based analysis reveals that the proposed ILP based approach (1) is very effective in reducing the energy consumptions of the applications tested and (2) generates much better energy savings than all the alternate schemes tested (including a unified scheme that combines voltage/frequency scaling and processor shutdown).

This work is supported in part by NSF Career Award #0093082 and by a grant from the GSRC.

Workload Clustering for Increasing Energy Savings on Embedded MPSoCs*

S.H.K. Narayanan,O. Ozturk,M. Kandemir
Department of Computer Science and Engineering
The Pennsylvania State University
{snarayan,ozturk,kandemir}@cse.psu.edu

M. Karakoy
180 Queen's Gate
Imperial College
mk22@doc.ic.ac.uk

ABSTRACT

Voltage/frequency scaling and processor low-power modes (i.e., processor shut-down) are two important mechanisms used for reducing energy consumption in embedded MPSoCs. While a unified scheme that combines these two mechanisms can achieve significant savings in some cases, such an approach is limited by the code parallelization strategy employed. In this paper, we propose a novel, integer linear programming (ILP) based workload clustering strategy across parallel processors, oriented towards maximizing the number of idle processors without impacting original execution times. These idle processors can then be switched to a low power mode to maximize energy savings, whereas the remaining ones can make use of voltage/frequency scaling. In order to check whether this approach brings any energy benefits over the pure voltage scaling based, pure processor shut-down based, or a simple unified scheme, we implemented four different approaches and tested them using a set of eight array/loop-intensive embedded applications. Our simulation-based analysis reveals that the proposed ILP based approach (1) is very effective in reducing the energy consumptions of the applications tested and (2) generates much better energy savings than all the alternate schemes tested (including a unified scheme that combines voltage/frequency scaling and processor shutdown).

I. INTRODUCTION

We can roughly divide the efforts on energy savings in embedded multi-processor system-on-a-chip architectures (MPSoCs) into two categories. In this first category are the studies that employ processor voltage/frequency scaling. The basic idea is to scale down voltage/frequency of a processor if its current workload is less than the workloads of other processors. In comparison, the studies in the second category shut down unused processors (i.e., put them into low-power states along with their private memory components) during the execution of the current computation. Both these techniques, i.e., voltage scaling and processor shut down, can be applied at the software level (e.g., directed by an optimizing compiler) or at the hardware-level (e.g., based on a past history-based workload/idleness detection algorithm). It is also conceivable to combine these two techniques under a unified optimizer.

Each of these techniques has its advantages and drawbacks. For example, a processor shut-down based scheme may not be applicable if there is no unused processor (note that this does not mean that the workloads of all the processors in the MPSoC are similar). Similarly, the effectiveness of a voltage scaling based scheme is limited by the number of voltage/frequency levels supported by the underlying hardware. In general, exploiting processor/memory shutdown saves more energy when it is applicable (as it reduces leakage energy significantly) or when we have only a couple of voltage/frequency levels to use. If this is not the case, then voltage scaling can be effective (and in some cases it is the only choice). Based on this discussion, one can expect a unified scheme to be successful. However, we want to re-iterate that if there is no unused (idle) processor in the current workload assignment, such a unified scheme simply reduces to a voltage scaling based approach.

Our goal in this paper is to explore a *workload (job) clustering* scheme that combines voltage scaling with processor shut-down[1]. The uniqueness of the proposed unified approach is that it maximizes the opportunities for processor shut-down by assigning workloads to processors carefully. It achieves this by clustering the original workloads of processors in as few processors as possible. In this paper, we discuss the technical details of this approach to energy saving in embedded MPSoCs. The proposed approach is ILP (integer linear programming) based; that is, it determines the optimal workload clusterings across the processors by formulating the problem using ILP and solving it using a linear solver. In order to check whether this approach brings any energy benefits over the pure voltage scaling based, pure processor shut-down based, or a simple unified scheme, we implemented four different approaches within our linear solver and tested them using a set of eight array/loop-intensive embedded applications. Our simulation-based analysis reveals that the proposed ILP based approach (1) is very effective in reducing the energy consumptions of the applications tested and (2) generates much better energy savings than all the alternate schemes tested (including one that combines voltage/frequency scaling and processor shutdown).

II. EMBEDDED MPSOC ARCHITECTURE, EXECUTION MODEL, AND RELATED WORK

The chip multiprocessor we consider in this work is a shared-memory architecture; that is, the entire address space is accessible by all processors. Each processor has a private L1 cache, and the shared memory is assumed to be off-chip. Optionally, we may include a (shared) L2 cache as well. Note that several architectures from academia and industry fit in this description [1, 10, 8, 9]. We keep the subsequent discussion simple by using a shared bus as the interconnect (though one could use fancier/higher bandwidth interconnects as well). We also use the MESI protocol (the choice is orthogonal to the focus of this paper) to keep the caches coherent across the CPUs. We assume that voltage level and frequency of each processor in this architecture can be set independently of the others, and also processors can be placed into low power modes independently. This paper focuses on a single-issue, five-stage (instruction fetch (IF), instruction decode/operand fetch (ID), execution (EXE), memory access (MEM), and write-back (WB) stages) pipelined datapath for each on-chip processor.

Our application execution model in this embedded MPSoC can be summarized as follows. We focus on array-based embedded applications that are constructed from loop nests. Typically, each loop nest in such an application is small but executes a large number of iterations and accesses/manipulates large datasets (typically multidimensional arrays of signals). We employ a loop nest based application parallelization strategy. More specifically, each loop nest is parallelized independently of the others. In this context, parallelizing a loop nest means distributing its iterations across processors and allowing processors to execute their portions in parallel. For example, a loop with 1000 iterations can be parallelized across 10 processors by allocating 100 itera-

*This work is supported in part by NSF Career Award #0093082 and by a grant from GSRC.

[1]In this paper, we use the terms "processor show-down" and "low-power mode" interchangeably.

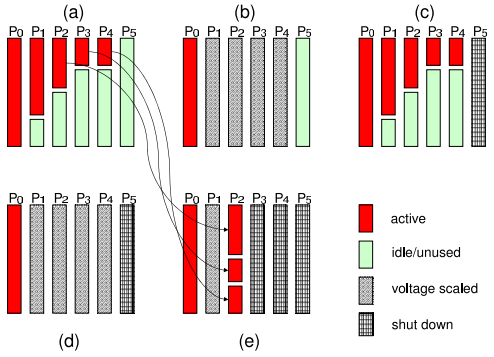

Figure 1: Comparison of different energy-saving approaches for a six processor architecture. Arrows indicate how the workloads (jobs) are clustered by our approach.

tions to each processor.

There are many proposals for power management of a dynamic voltage scaling-capable processor. Most of them are at the operating system level and are either task-based [12] or interval-based [4]. While some proposals aim at reducing energy without compromising performance, a recent study by Grunwald et al [5] observed noticeable performance loss for some interval-based algorithms using actual measurements. Most of the existing compiler based studies such as [11] target single processor architectures. In comparison, our work targets at a chip multiprocessor based environment and combines voltage scaling and processor shutdown. [15] presents and analyzes a voltage/frequency scaling scheme but, they do not consider processor shut-down. [6] employs processor a shut-down based mechanism but does not consider voltage/frequency scaling. In our experimental evaluation, we compare our approach to pure voltage/frequency scaling and to pure processor shut-down as well.

III. OUR APPROACH

III.1 Overview

Figure 1 compares four different alternate schemes that saves energy in an embedded MPSoC architecture. It is assumed, for illustrative purposes, that the architecture has six processors. In Figure 1(a) shows the workloads of the processors (i.e., the jobs assigned to them) in a given loop nest. These are assumed to be the loads either estimated by the compiler or calculated through profiling and are for a single nest. Figures 1(b) and (c) show the scenarios with pure voltage/frequency scaling and pure processor shut-down based approaches, respectively. In (b), four out of our six processors take advantage of voltage scaling (note that P_5 is not used in the computation at all). In (c), on the other hand, we can place only one processors (P_5) into the low-power mode. A combination of these two approaches is depicted in Figure 1(d). Basically, this version combines the benefits of voltage/frequency scaling and processor shut-down. Finally, the result that can be obtained by the ILP approach proposed in this paper is illustrated in Figure 1(e). Note that what our approach essentially does is to *cluster* the total amount of computational load in as fewer processors as possible so that the number of unused processors is maximized. In this particular case, the original loads of three processors (P_2, P_3, and P_4) are combined and assigned to processor P_2. As a result, processors P_3 and P_4 can also be placed into the low-power mode (along with their private memory components) to maximize energy savings, in addition to P_5. The next subsection gives the technical details of this approach. When there are opportunities, our approach can also use voltage/frequency scaling for the clustered jobs. It is important to point out that the benefits from our approach can be expected to be even more significant when the number of voltage/frequency levels is small. In such a case, a pure voltage/frequency scaling based approach cannot stretch the execution time of a processor to fill the available slack completely.

However, we first need to clarify two important issues. Someone may ask at this point "why has the application (corresponding to the scenario in Figure 1(a)) not been parallelized at the first place as shown in Figure 1(e)?" There are several reasons for this. First, most current code parallelizers do not consider any energy optimizations. Therefore, there is really little reason for calculating the workloads of individual processors, and thus little opportunity for workload clustering. Second, the conventional parallelizing compilers try to use as many processors as possible for executing a given computation unless there exists a compelling reason to do otherwise (e.g., the excessive synchronization costs). Third, in many cases, trying to cluster computation in very few processors can have an impact on execution cycles. Since most parallelizing compilers do not predict or quantify this impact, they do not attempt such clusterings, being on the conservative side.

The second issue is that, it is possible that the scenario depicted in Figure 1(e) has poor data locality as compared to scenarios in Figures 1(b), (c), and (d). This is because conventional code parallelizers generally try to achieve good data locality, by ensuring that each processor mostly uses the same set of data elements as much as possible (i.e., high data reuse). As a result, the scenario in Figure 1(e) can lead to an increase in data cache misses, which in turn increases overall energy consumption. This overhead should also be factored in our clustering approach to ensure a fair comparison.

The main contribution of the ILP approach proposed in this paper is to obtain, for each loop nest in an application, the result shown in Figure 1(e), given the initial scenario (workload assignment) shown in Figure 1(a) and thus reduce energy consumption.

III.2 Technical Details and Problem Formulation

This section elaborates on the ILP model used to represent the problem. In our problem, there exist a set of jobs (workloads) that have to be executed on a set of available processors in the embedded MPSoC such that the total energy spent by the system is minimal and that the execution of the jobs completes within a specified time limit, T_{max}.[2] The processors can run different jobs at different voltage and frequency levels, which affects energy consumption. The energy expended by each processor is the sum of the dynamic energy as well as the leakage energy expended while running. The rest of this section describes the ILP model in detail.

III.2.1 System and Job Model

We assume that the jobs are members of the set J consisting of J_{max} elements and the processors belong to the set P in which there are P_{max} elements. The processors can run at V_{num} discrete set of voltage/frequency levels (as supported by the architecture). It is assumed that only one job can run on a processor at anytime and that once a job starts running on a processor, it runs uninterrupted to completion. However, a processor can be assigned to run more than one job, as a result of workload clustering. The duration that the job occupies the processor is dependent on the supply voltage/frequency as well as the the frequency at which the processor is running that particular job. The time (latency) each job takes up at different voltage levels is specified in the array $Job\_Length(j, v)$. Similarly, the dynamic energy spent by each job at different voltage levels varies and is captured by $Job\_Dynamic(j, v)$.[3] $Total\_Energy$ is the sum of the energies spent by all jobs on all processors due to their running as well as the leakage energy consumed by the processors. This is the metric whose value we want to minimize.

III.2.2 Mathematical Programming Model

The constraints specified below give the mathematical representation of our model. We use 0-1 integer linear programming (ILP). This ILP formulation is executed for each loop nest separately. Table 1 gives the notation used in our formulation.

Job Assignment Constraints. The 0-1 variable $X(p, j, v)$ determines whether processor p runs job j at voltage/frequency level v. One

[2]In this paper, we do not assume a specific code (loop nest) parallelization strategy. Rather, we assume that each loop nest is parallelized using one of the known techniques. For each loop nest, T_{max} is determined by the processor with the largest workload. This is to ensure that our workload clustering does not have a negative impact on execution times.

[3]Here j represents a job (workload) and v represents a voltage (frequency) level. In our implementation, the entries of $Job\_Length(j, v)$ and $Job\_Dynamic(j, v)$ are filled using profiling. All energy estimations are performed using Wattch [2] under the 70nm process technology. The increase in data cache misses as a result of clustering is captured during our profiling.

Notation	Explanation
$Job\_Dynamic(j,v)$	Dynamic energy for running job (workload) j at voltage v
$Job\_Length(j,v)$	Time taken to run job j at voltage v
$X(p,j,v)$	Value is 1 if job j runs on processor p at voltage v
J	Set of jobs
P	Set of processors
T_max	Time deadline before which all jobs must finish
J_max	Total number of jobs to be executed
P_max	Total number of processors available
V_num	Total number of voltage (and frequency) levels available
$Total\_Energy$	Total energy consumption of the system (to be minimized)
$Leakage\_Value$	Leakage energy spent by a processor if it is not shut down

Table 1: Notation used in our model.

job runs completely on one processor and all jobs are scheduled to run only once. This is specified as follows:

$$\forall p \in P \quad \forall j \in J \quad \forall v \in V \quad X(p,j,v) \in \{0|1\} \tag{1}$$

$$\forall j \in J \sum_{p=0}^{P_{max}-1} \sum_{v=0}^{V_{num}-1} X(p,j,v) = 1 \tag{2}$$

Constraint (1) expresses the term $X(j,p,v)$ as a binary variable; a processor either runs the job or it does not. Constraint (2) states that each job can be run only on one processor and that all jobs are assigned to some processors (i.e., no job is left unassigned). Notice that we want to determine the value of $X(p,j,v)$ for all p, j, and v.

Deadline Constraints. Jobs are assigned to processors as long as they can meet the time deadline that is specified. Constraint (3) expresses this:

$$\forall p \in P \sum_{j=0}^{J_{max}-1} \sum_{v=0}^{V_{num}-1} X(p,j,v) * Job\_Length(j,v) \leq T_{max} \tag{3}$$

Note that T_{max} is determined, for each loop nest, by the longest (largest) workload.

Clustering and Processor Shut-Down Constraints. Multiple jobs are run on the same processor if the number of jobs, J_{max}, exceeds the number of processor, P_{max}, but also if such an arrangement reduces the overall energy spent by the system. In case a processor is not assigned any job, either because of clustering of jobs or because $J_{max} < P_{max}$ or because of both these reasons, then it is shut down. Such a processor does not consume any dynamic energy as it has no jobs running on it and it does not consume any leakage energy since it is shut down (except for some small amount of leakage in memory components). Constraint (4) is introduced to capture processor shutdown:

$$\forall p \in P, \forall j \in J, \forall v \in V \quad Busy(p) \geq X(p,j,v) \tag{4}$$

For a particular processor p, $Busy(p)$ is necessarily 1 if any of the values in $X(p,j,v)$ is 1. Through this constraint, the value of $Busy(P)$ is not explicitly expressed if all values in $X(p,j,v)$ are 0. However, a value of 1 in $Busy(p)$ adds leakage to the overall energy. As the objective of the ILP-based model is to reduce energy, $Busy(p)$ will be assigned to be 0 if all values in $X(p,j,v)$ are 0.[4]

Leakage and Dynamic Energy Calculation. The following expressions capture the leakage energy and dynamic energy spent by the system as the sum of the leakage and dynamic energies, respectively, spent by each processor. The total amount of dynamic energy spent by a processor is the sum of the dynamic energies spent for each job that is run on that processor. This is captured by Expression (5):

$$D\_Energy = \sum_{p=0}^{P_{max}-1} \sum_{j=0}^{J_{max}-1} \sum_{v=0}^{V_{num}-1} X(p,j,v) * Job\_Energy(j,v) \tag{5}$$

Expression (6) calculates the leakage energy spent. As mentioned earlier, if $Busy(p)$ is 1, then leakage is spent by processor p.

$$L\_Energy = Leakage\_Value * \sum_{p=0}^{P_{max}-1} * Busy(p) \tag{6}$$

Objective Function. The objective function which is the total energy spent by the system is the sum of the the leakage and dynamic energies. This is the objective function that our approach tries to minimize:

$$Total\_Energy = D\_Energy + L\_Energy \tag{7}$$

The constraints and expressions mentioned in this section are sufficient to express our problem within ILP. We next look at the additional constraints that can be used in order to handle two special cases.

Voltage/Frequency Scaling without Clustering. To model classical voltage/frequency scaling within our ILP formulation, an input value $Assign(j,p)$ should specify the processor on which each job runs. Further, by connecting this value to that of $X(j,p,v)$, all jobs are forced to run on the assigned processors alone. This connection can be captured by the following constraint:

$$\forall p \in P, \forall j \in J \sum_{v=0}^{V_{num}-1} X(p,j,v) = Assign(p,j) \tag{8}$$

Clustering without Voltage/Frequency Scaling. To model job clustering without voltage and frequency scaling, we need to constrain the choice of available voltage frequency levels to either each processor individually or all processors. In the case of constraining the voltage levels of all processors to one value, Constraint (9) can be used to ensure that no jobs are assigned voltage levels other than the one specified.

$$\forall p \in P, \forall j \in J, \forall v \in V - \{v'\} \quad X(p,j,v) = 0. \tag{9}$$

To constrain each individual processor to an independent voltage level, Constraint (10) below can be used.

$$\forall p \in P, \forall j \in J, \forall v \in V - \{v'_p\} \quad X(p,j,v) = 0. \tag{10}$$

Here, v' and v'_p are the universal and individual (for processor p) voltage levels, respectively. These constraints simply limit the voltage levels to be used. In this case, the decision to cluster jobs together on a processor is made by our solver and depends on whether it results in a lowered overall energy consumption.

IV. EXPERIMENTAL EVALUATION

We present only energy results in this section. The reason is that none of the techniques evaluated increases original execution cycles (i.e., we do not exceed T_{max} in any loop nest). Specifically, for each loop nest, the processor with the largest workload sets the limit for voltage/frequency scaling and processor shut-down. The ILP solver used in our experiments is lp_solve [7]. We observed that the ILP solution times with the application codes in our experimental suite varied between 56.7 seconds and 13.2 minutes. Considering the large energy savings, these solution times are within tolerable limits.

All the experimental results are obtained using the SIMICS simulation platform [13]. Specifically, we embedded in the SIMICS platform timing and energy models that help us simulate the behavior of the following four schemes: VS (pure voltage/frequency scaling based approach); SD (pure processor shut-down based approach); VS+SD (a unified approach that combines VS and SD); and CLUSTERING (the ILP-based approach proposed in this paper). The default simulation parameters used in our experiments are listed in Table 2. In the last three schemes, when a processor is unused in the current loop nest, it

[4]To preserve data in memory components, a shut-down processor consumes some leakage [3]. Our experiments are performed based on this principle. However, in our presentation of the ILP formulation, we assume no leakage consumption in the shut-down state for ease of presentation.

Simulation Parameter	Value
Processor Speed	400MHz
Number of Processors	8
Lowest/Highest Voltage Levels	0.8V/1.4V
Number of Voltage Levels	4
Instruction Cache	8KB 2-way associative 32 byte blocks
Data Cache	8KB 2-way associative 32 byte blocks
Memory	32MB (banked)
Off-Chip Memory Access Latency	100 cycles
Bus Arbitration Delay	5 cycles
Replacement Policy	Strict LRU
Cache Dynamic Energy Consumption	0.6 nJ
Memory Dynamic Energy Consumption	1.17 nJ
Leakage Energy Consumption for 32 bytes Normal Operation Shut-Down State	 4.49 pJ 0.92 pJ
Resynchronization Time for Shut-Down State	30 msec
Resynchronization Time for Voltage Scaling	5 msec

Table 2: The default simulation parameters.

is shut-down and its L1 instruction and data caches are placed into the low-power mode. The specific low-power mode employed in this paper is from [3].

We used 8 array/loop-intensive applications for evaluating the four approaches mentioned above: 3D, DFE, LU, SPLAT, MGRID, WAVE5, SPARSE, and XSEL. 3D is an image-based modeling application that simplifies the task of building 3D models and scenes. DFE is a digital image filtering and enhancement code. LU is an LU decomposition program. SPLAT is a volume rendering application which is used in multi-resolution volume visualization through hierarchical wavelet splitting. MGRID and WAVE5 are C versions of two Spec95FP applications. SPARSE is an image processing code that performs sparse matrix operations, and finally, XSEL is an image rendering code. These C programs are written in such a fashion that they can operate on inputs of different sizes. We first ran these applications through our simulator without any voltage scaling or processor shut-down. This version of an application is referred to as the *base version* or the *base execution* in the remainder of this paper. The energy consumptions (which include energies spent in processors, caches, interconnects, and off-chip memory) under the base execution are 272.1mJ, 388.3mJ, 197.9mJ, 208.4mJ, 571.0mJ, 466.2mJ, 292.2mJ, and 401.5mJ for 3D, DFE, LU, SPLAT, MGRID, WAVE5, SPARSE, and XSEL, respectively. The energy results presented in this section are given as *normalized* values with respect to this base execution.

To calculate the dynamic energy consumptions for caches and memory, we used the Cacti tool [14]. We approximated the leakage energy consumption by assuming that the leakage energy per cycle for 4KB SRAM is equal to the dynamic energy consumed per access to a 32 byte data from the same SRAM. Note that this assumption tries to capture the anticipated importance of leakage energy in the future as leakage becomes the dominant part of energy consumption for 0.10 micron (and below) technologies for the typical internal junction temperatures in a chip. In the shut-down state, a processor and its caches consume only a small percentage of their original (per cycle) leakage energy. However, when a processor and its data and instruction caches in the shut-down state are needed, they need to be reactivated (resynchronized). This resynchronization costs extra execution cycles as well as extra energy consumption as noted in [3], and *all these costs are captured in our simulations and included in all our results.*

Our first set of results, the normalized energy consumptions with the different schemes, are presented in Figure 2. Each group of bars in this graph correspond to an application, and the last group of bars gives the average results across all eight applications. The energy savings achieved by the VS scheme are not very large (6.55% on the average). There are two main reasons for this. The first one is the inherent characteristics of some applications. More specifically, when there are no long idle periods, VS is not applicable. The second reason is the limited number of voltage/frequency levels used in the default configuration (see Table 2). In comparison, the SD scheme behaves in a different

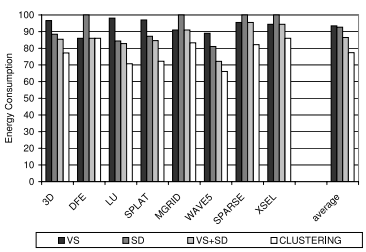

Figure 2: Normalized energy consumptions.

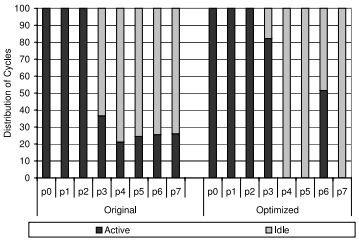

Figure 3: The active and idle periods of processors in the mx3-raw.c routine from MGRID.

manner. While it is not applicable in some cases (e.g., in applications DFE, MGRID, SPARSE, and XSEL), the energy savings it brings are significant in cases where it is applicable. VS+SD simply combines the benefits of the VS and SD schemes, reducing to VS when SD is not applicable. The average energy savings (across all eight applications) achieved by SD and VS+SD are 7.36% and 13.52%, respectively. The highest energy savings are obtained by our ILP-based approach, which is 22.65% on the average. These results clearly show the potential benefits of our ILP-based workload clustering approach.

To better illustrate where our energy benefits are coming from, we give in Figure 3 the percentage of time each processor spends in the active and idle states for procedure mx3-raw.c, one of the thirteen subprograms in application MGRID. We see from this graph that our ILP-based approach is able to increase the number of idle processors. We observed similar trends with most of other procedures in our applications. These results explain the energy benefits observed in Figure 2.

V. CONCLUSIONS

This paper proposes a workload clustering scheme for embedded MPSoCs that combines voltage scaling with processor shut-down. The uniqueness of the proposed unified approach is that it maximizes the use of processor shut-down by clustering workloads (jobs) in as few processors as possible. We tested this approach along with three alternate schemes using a simulation-based platform and eight embedded applications. Our experiments show that this clustering approach is very effective in reducing energy consumption and generates better results than the three alternative schemes evaluated.

VI. REFERENCES

[1] L. A. Barroso et al. Piranha: A Scalable Architecture Based on Single-Chip Multiprocessing. *Proceedings of ISCA'2000.*

[2] D. Brooks et al. Wattch: a framework for architectural-level power analysis and optimizations In *Proceedings of ISCA,* Canada 2000.

[3] K. Flautner, N. Kim, S. Martin, D. Blaauw, and T. Mudge. Drowsy Caches: Simple techniques for reducing leakage power. *Proceedings of ISCA,* 2002.

[4] K. Govil, E. Chan, and H. Wasserman. Comparing Algorithms for Dynamic Speed-Setting of a Low-Power CPU. *Proceedings of the 1st ACM International Conference on Mobile Computing and Networking,* November 1995.

[5] D. Grunwald, P. Levis, K. Farkas, C. Morrey III, and M. Neufeld. Policies for Dynamic Clock Scheduling. *Proceedings OSDI'2000.*

[6] I. Kadayif, M. Kandemir, and U. Sezer. An Integer Linear Programming Based Approach for Parallelizing Applications in On-Chip Multiprocessors. In *Proceedings of DAC'2002.*

[7] lp_solve. ftp://ftp.es.ele.tue.nl/pub/lp_solve/

[8] MAJC-5200. http://www.sun.com/microelectronics/ MAJC/5200wp.html

[9] MP98: A Mobile Processor. http://www.labs.nec.co.jp/MP98/top-e.htm.

[10] K. Olukotun, B. A. Nayfeh, L. Hammond, K. Wilson, and K. Chang. The Case for a Single Chip Multiprocessor. *Proceedings of ASPLOS'1996.*

[11] H. Saputra et al. Energy-Conscious Compilation Based on Voltage Scaling. *Proceedings of ACM SIGPLAN Joint Conference LCTES'02 and SCOPES'02,* Berlin , Germany, June, 2002.

[12] Y. Shin, K. Choi, and T. Sakurai. Power Optimization of Real-Time Embedded Systems on Variable Speed Processors. *Proceedings of the International Conference on Computer-Aided Design,* November 2000.

[13] SIMICS. http://www.virtutech.com/simics/simics.html.

[14] S. Wilton and N. Jouppi. Cacti: An enhanced cache access and cycle time model. *IEEE Journal of Solid-State Circuits,* May 1996.

[15] Q. Wu, P. Juang, M. Martonosi, and D. W. Clark. Formal on-line methods for voltage/frequency control in multiple clock domain microprocessors. *Proceedings of ASPLOS'2004.*